Exhibit 7

EXHIBIT 7

CYWEE GROUP LTD,
vs.
HUAWEI DEVICE CO. LTD.,
HUAWEI DEVICE (DONGGUAN) CO. LTD., AND
HUAWEI DEVICE USA, INC.

UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION

EXEMPLARY CLAIM CHART

U.S. PATENT NO. 8,552,978 – Huawei Honor 8 Infringement Contentions

These contentions are disclosed to only provide notice of Plaintiff's theories of infringement. These contentions do not constitute proof nor do they marshal Plaintiff's evidence of infringement to be presented during trial.

Claim 10

Claim 10 with claim constructions (text in brackets [] reflects the Court's claim construction or the parties' agreed claim construction in *CyWee Group, Ltd. v. Apple Inc.*, No. 3:13-cv-01853-HSG). Construed terms and constructions are underlined.

10. A method for compensating rotations of a 3D pointing device, comprising:

generating an orientation output associated with an orientation of the 3D pointing device associated with three coordinate axes of a global reference frame associated with Earth;

generatinq a first signal set comprising axial accelerations associated with movements and rotations of the 3D pointing device in the spatial reference frame;

generating a second signal set associated with Earth's magnetism; generating the orientation output based on the first signal set, the second signal set and the rotation output or based on the first signal set and the second signal set;

generating a rotation output associated with a rotation of the 3D pointing device associated with three coordinate axes of a spatial reference frame associated with the 3D pointing device; and

<u>using the orientation output and the rotation output to generate a transformed output associated with a fixed reference frame associated with a display device [Court's construction: using the orientation output and the rotation output to generate a transformed output that corresponds to a two-dimensional movement in a plane that is parallel to the screen of a display device]</u>, wherein the orientation output and the rotation output is generated by a nine-axis motion sensor module; obtaining one or more resultant deviation including a plurality of deviation angles using a plurality of measured magnetisms Mx, My, Mz and a plurality of predicted magnetism Mx′, My′ and Mz′ for the second signal set.

Claim 10
A method for compensating rotations of a 3D pointing device, comprising:
 Huawei Honor 8

Claim 10

generating an **orientation output** associated with an orientation of the 3D pointing device associated with three coordinate axes of a **global reference frame associated with Earth**;

When the orientation sensor is software-based, the **orientation output** is the attitude of the device that can be represented by the azimuth, pitch, and roll angles relative to the magnetic North Pole associated with a **global reference frame associated with Earth**.

Rotation vector

Underlying physical sensors: Accelerometer, Magnetometer, and Gyroscope

Reporting-mode: *Continuous*

`getDefaultSensor(SENSOR_TYPE_ROTATION_VECTOR)` *returns a non-wake-up sensor*

A rotation vector sensor reports the orientation of the device relative to the East-North-Up coordinates frame. It is usually obtained by integration of accelerometer, gyroscope, and magnetometer readings. The East-North-Up coordinate system is defined as a direct orthonormal basis where:

- X points east and is tangential to the ground.
- Y points north and is tangential to the ground.
- Z points towards the sky and is perpendicular to the ground.

The orientation of the phone is represented by the rotation necessary to align the East-North-Up coordinates with the phone's coordinates. That is, applying the rotation to the world frame (X,Y,Z) would align them with the phone coordinates (x,y,z).

Source: https://source.android.com/devices/sensors/sensor-types#rotation_vector

Claim 10

generating a **first signal set** comprising axial accelerations associated with movements and rotations of the 3D pointing device in the **spatial reference frame**;

Accelerometer

Reporting-mode: *Continuous*

`getDefaultSensor(SENSOR_TYPE_ACCELEROMETER)` *returns a non-wake-up sensor*

An accelerometer sensor reports the acceleration of the device along the 3 sensor axes. The measured acceleration includes both the physical acceleration (change of velocity) and the gravity. The measurement is reported in the x, y and z fields of sensors_event_t.acceleration.

All values are in SI units (m/s^2) and measure the acceleration of the device minus the force of gravity along the 3 sensor axes.

Source: https://source.android.com/devices/sensors/sensor-types#accelerometer

Sensor Coordinate System

In general, the sensor framework uses a standard 3-axis coordinate system to express data values. For most sensors, the coordinate system is defined relative to the device's screen when the device is held in its default orientation (see figure 1). When a device is held in its default orientation, the X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points toward the outside of the screen face. In this system, coordinates behind the screen have negative Z values. This coordinate system is used by the following sensors:

- Acceleration sensor
- Gravity sensor
- Gyroscope
- Linear acceleration sensor
- Geomagnetic field sensor

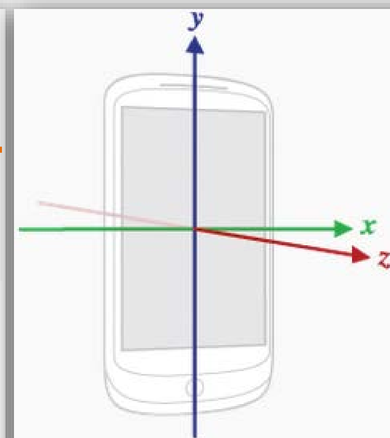


Figure 1. Coordinate system (relative to a device) that's used by the Sensor API.

Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

U.S. Patent No. 8,552,978 – Huawei Honor 8

Claim 10
generating a **second signal set** associated with **Earth's magnetism**;
The magnetometer (i.e., the compass) generates a second signal set associated with **Earth's magnetism**. Magnetic field sensor Reporting-mode: *Continuous* `getDefaultSensor(SENSOR_TYPE_MAGNETIC_FIELD)` *returns a non-wake-up sensor* `SENSOR_TYPE_GEOMAGNETIC_FIELD == SENSOR_TYPE_MAGNETIC_FIELD` A magnetic field sensor (also known as magnetometer) reports the ambient magnetic field, as measured along the 3 sensor axes. The measurement is reported in the x, y and z fields of `sensors_event_t.magnetic` and all values are in micro-Tesla (uT). **Source**: https://source.android.com/devices/sensors/sensor-types#magnetic_field_sensor

Claim 10

generating the **orientation output** based on the **first signal set**, the **second signal set** and the **rotation output** or based on the **first signal set** and the **second signal set**;

The Android source code shows generating the **orientation output** based on the **first signal set**, the **second signal set** and the **rotation output**.

The `handleGyro()` function passes **rotation output** `w` to the `predict()` function and the `update()` function to calculate an **orientation output**, `x0`.

```
void Fusion::handleGyro(const vec3_t& w, float dT) {
    if (!checkInitComplete(GYRO, w, dT))
        return;
```

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
```

```
    x0 = O*q;
```

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
```

```
    const vec3_t e(z - Bb);
    const vec3_t dq(K[0]*e);

    q += getF(q)*(0.5f*dq);
    x0 = normalize_quat(q);
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 10

generating the **orientation output** based on the **first signal set**, the second signal set and the rotation output or based on the **first signal set** and the second signal set;

The `handleAcc()` function passes the accelerometer measurements (**first signal set**) `a` to the `update()` function, which updates the **orientation output** `x0`.

```
status_t Fusion::handleAcc(const vec3_t& a, float dT) {
    if (!checkInitComplete(ACC, a, dT))
        return BAD_VALUE;
```

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
```

```
    const vec3_t e(z - Bb);
    const vec3_t dq(K[0]*e);

    q += getF(q)*(0.5f*dq);
    x0 = normalize_quat(q);
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 10

generating the **orientation output** based on the first signal set, the **second signal set** and the rotation output or based on the first signal set and the **second signal set**;

The `handleMag()` function passes the magnetometer measurements (**second signal set**) `m` to the same `update()`, which also updates the **orientation output** `x0`.

```
status_t Fusion::handleMag(const vec3_t& m) {
    if (!checkInitComplete(MAG, m))
        return BAD_VALUE;
```

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
```

```
    const vec3_t e(z - Bb);
    const vec3_t dq(K[0]*e);

    q += getF(q)*(0.5f*dq);
    x0 = normalize_quat(q);
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 10

generating a **rotation output** associated with a rotation of the 3D pointing device associated with three coordinate axes of a **spatial reference frame** associated with the 3D pointing device; and

Gyroscope

Reporting-mode: *Continuous*

`getDefaultSensor(SENSOR_TYPE_GYROSCOPE)` *returns a non-wake-up sensor*

A gyroscope sensor reports the rate of rotation of the device around the 3 sensor axes.

Rotation is positive in the counterclockwise direction (right-hand rule). That is, an observer looking from some positive location on the x, y or z axis at a device positioned on the origin would report positive rotation if the device appeared to be rotating counter clockwise. Note that this is the standard mathematical definition of positive rotation and does not agree with the aerospace definition of roll.

Source: https://source.android.com/devices/sensors/sensor-types#gyroscope

Sensor Coordinate System

In general, the sensor framework uses a standard 3-axis coordinate system to express data values. For most sensors, the coordinate system is defined relative to the device's screen when the device is held in its default orientation (see figure 1). When a device is held in its default orientation, the X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points toward the outside of the screen face. In this system, coordinates behind the screen have negative Z values. This coordinate system is used by the following sensors:

- Acceleration sensor
- Gravity sensor
- Gyroscope
- Linear acceleration sensor
- Geomagnetic field sensor

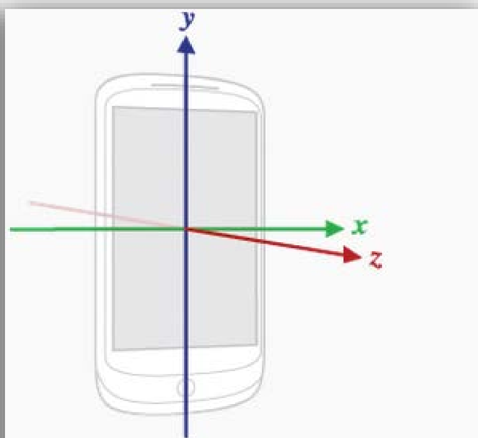


Figure 1. Coordinate system (relative to a device) that's used by the Sensor API.

Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

Claim 10
using the orientation output and the rotation output to generate a transformed output associated with a **fixed reference frame** associated with a **display device** [<u>Court's construction: using the orientation output and the rotation output to generate a transformed output that corresponds to a two-dimensional movement in a plane that is parallel to the screen of a display device],</u>
The **fixed reference frame** is defined by the horizontal and vertical axes of pixels on Huawei Honor 8's **display device**.

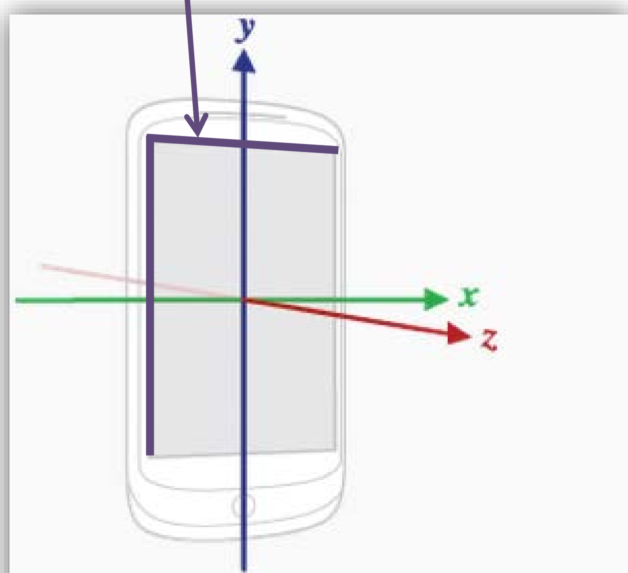


Figure 1. Coordinate system (relative to a device) that's used by the Sensor API.

Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

Claim 10

using the orientation output and the rotation output to generate a transformed output associated with a **fixed reference frame** associated with a **display device** [Court's construction: using the orientation output and the rotation output to generate a transformed output that corresponds to a two-dimensional movement in a plane that is parallel to the screen of a display device],

The `remapCoordinateSystem()` function transforms the orientation output (`inR`) to a transformed output (`outR`), associated with a two dimensional movement in a plane that is parallel to the screen of a **display device**.

```
    public static boolean remapCoordinateSystem(float[] inR, int X, int Y, float[] outR) {
        if (inR == outR) {
            final float[] temp = sTempMatrix;
            synchronized (temp) {
                // we don't expect to have a lot of contention
                if (remapCoordinateSystemImpl(inR, X, Y, temp)) {
                    final int size = outR.length;
                    for (int i = 0; i < size; i++) {
                        outR[i] = temp[i];
                    }
                    return true;
                }
            }
        }
        return remapCoordinateSystemImpl(inR, X, Y, outR);
    }
```

Source: https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/hardware/SensorManager.java

```
boolean remapCoordinateSystem (float[] inR,
                int X,
                int Y,
                float[] outR)
```

Rotates the supplied rotation matrix so it is expressed in a different coordinate system. This is typically used when an application needs to compute the three orientation angles of the device (see `getOrientation(float[], float[])`) in a different coordinate system.

When the rotation matrix is used for drawing (for instance with OpenGL ES), it usually **doesn't need** to be transformed by this function, unless the screen is physically rotated, in which case you can use `Display.getRotation()` to retrieve the current rotation of the screen. Note that because the user is generally free to rotate their screen, you often should consider the rotation in deciding the parameters to use here.

Source: https://developer.android.com/reference/android/hardware/SensorManager.html#remapCoordinateSystem(float[], int, int, float[])

Claim 10

wherein the orientation output and the rotation output is generated by a **nine-axis motion sensor module**;

The Huawei Honor 8 includes a 3-axis gyroscope, a 3-axis accelerometer, and a 3-axis magnetometer which form a **nine-axis motion sensor module.**

Claim 10
obtaining one or more resultant deviation including a plurality of deviation angles using a plurality of **measured magnetisms Mx, My, Mz** and a plurality of predicted magnetism Mx′, My′ and Mz′ for the second signal set.
The **measured magnetisms Mx, My, Mz** are `values[0]-[2]`. `Sensor.TYPE_MAGNETIC_FIELD_UNCALIBRATED`: Similar to `TYPE_MAGNETIC_FIELD`, but the hard iron calibration is reported separately instead of being included in the measurement. Factory calibration and temperature compensation will still be applied to the "uncalibrated" measurement. Assumptions that the magnetic field is due to the Earth's poles is avoided. The values array is shown below: • values[0] = x_uncalib • values[1] = y_uncalib • values[2] = z_uncalib • values[3] = x_bias • values[4] = y_bias • values[5] = z_bias x_uncalib, y_uncalib, z_uncalib are the measured magnetic field in X, Y, Z axes. Soft iron and temperature calibrations are applied. But the hard iron calibration is not applied. The values are in micro-Tesla (uT). x_bias, y_bias, z_bias give the iron bias estimated in X, Y, Z axes. Each field is a component of the estimated hard iron calibration. The values are in micro-Tesla (uT). Hard iron - These distortions arise due to the magnetized iron, steel or permanenet magnets on the device. Soft iron - These distortions arise due to the interaction with the earth's magentic field. **Source**: http://developer.android.com/reference/android/hardware/SensorEvent.html#values

Claim 10

obtaining one or more resultant deviation including a plurality of deviation angles using a plurality of **measured magnetisms Mx, My, Mz** and a plurality of **predicted magnetism** Mx′, My′ and Mz′ for the second signal set.

The **measured magnetisms**, z, and a **predicted magnetism**, Bb, are used to calculate a global variable x0 in quaternion form.

measured magnetisms

```
        update(m, Bm, mParam.magStdev);
```

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    const vec3_t Bb(A*Bi);
```

```
    const vec3_t e(z - Bb);
    const vec3_t dq(K[0]*e);

    q += getF(q)*(0.5f*dq);
    x0 = normalize_quat(q);
```

predicted magnetism

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 10

obtaining one or more **resultant deviation including a plurality of deviation angles** using a plurality of measured magnetisms Mx, My, Mz and a plurality of predicted magnetism Mx′, My′ and Mz′ for the second signal set. .

The global variable `x0` is in quaternion form, and can easily be converted to resultant angles.

According to Android's developer library, the `getOrientation()` function "computes the device's orientation based on the rotation matrix," and returns **deviation angles** including the Azimuth, Pitch, and Roll angles.

getOrientation

Added in API level 3

```
float[] getOrientation (float[] R,
                float[] values)
```

Computes the device's orientation based on the rotation matrix.

When it returns, the array values are as follows:

- values[0]: *Azimuth*, angle of rotation about the -z axis. This value represents the angle between the device's y axis and the magnetic north pole. When facing north, this angle is 0, when facing south, this angle is π. Likewise, when facing east, this angle is π/2, and when facing west, this angle is -π/2. The range of values is -π to π.
- values[1]: *Pitch*, angle of rotation about the x axis. This value represents the angle between a plane parallel to the device's screen and a plane parallel to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the top edge of the device toward the ground creates a positive pitch angle. The range of values is -π to π.
- values[2]: *Roll*, angle of rotation about the y axis. This value represents the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the left edge of the device toward the ground creates a positive roll angle. The range of values is -π/2 to π/2.

The `getRotationMatrixFromVector()` function "convert[s] a rotation vector to a rotation matrix," and the `getQuaternionFromVector()` function "convert[s] a rotation vector to a normalized quaternion." Therefore, the quaternion, `x0`, can be easily converted to its mathematically equivalent form, rotation matrix, and used by `getOrientation()` function to compute the orientation in its angular form.

Source: https://developer.android.com/reference/android/hardware/SensorManager.html#getOrientation(float[], float[])

Claim 10

obtaining one or more **resultant deviation including a plurality of deviation angles** using a plurality of measured magnetisms Mx, My, Mz and a plurality of predicted magnetism Mx′, My′ and Mz′ for the second signal set. .

Rotation vector

Underlying physical sensors: Accelerometer, Magnetometer, and Gyroscope

Reporting-mode: Continuous

`getDefaultSensor(SENSOR_TYPE_ROTATION_VECTOR)` *returns a non-wake-up sensor*

Source: https://source.android.com/devices/sensors/sensor-types#rotation_vector

getRotationMatrixFromVector

added in API level 9

```
void getRotationMatrixFromVector (float[] R,
                float[] rotationVector)
```

Helper function to convert a rotation vector to a rotation matrix. Given a rotation vector (presumably from a ROTATION_VECTOR sensor), returns a 9 or 16 element rotation matrix in the array R. R must have length 9 or 16. If R.length == 9, the following matrix is returned:

Source: https://developer.android.com/reference/android/hardware/SensorManager.html#getRotationMatrixFromVector(float[], float[])

getOrientation

added in API level 3

```
float[] getOrientation (float[] R,
                float[] values)
```

Computes the device's orientation based on the rotation matrix.

When it returns, the array values are as follows:

- values[0]: *Azimuth*, angle of rotation about the -z axis. This value represents the angle between the device's y axis and the magnetic north pole. When facing north, this angle is 0, when facing south, this angle is π. Likewise, when facing east, this angle is π/2, and when facing west, this angle is -π/2. The range of values is -π to π.
- values[1]: *Pitch*, angle of rotation about the x axis. This value represents the angle between a plane parallel to the device's screen and a plane parallel to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the top edge of the device toward the ground creates a positive pitch angle. The range of values is -π to π.
- values[2]: *Roll*, angle of rotation about the y axis. This value represents the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the left edge of the device toward the ground creates a positive roll angle. The range of values is -π/2 to π/2.

Source: https://developer.android.com/reference/android/hardware/SensorManager.html#getOrientation(float[], float[])

Claim 12

The method of claim 10, wherein the **orientation output** is a **rotation matrix**, a quaternion, a rotation vector, or comprises three orientation angles.

Several methods are provided for receiving the **orientation output** as a rotation matrix, a quaternion, or three orientation angles. The `getRotationMatrix()` function outputs a **rotation matrix**.

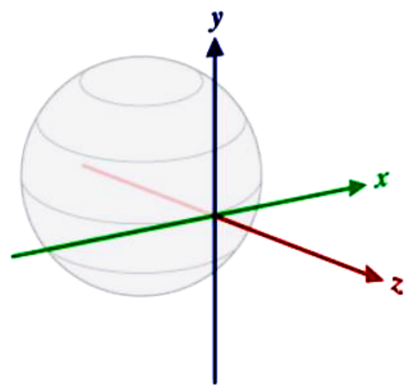
getRotationMatrix

added in API level 3

```
boolean getRotationMatrix (float[] R,
                float[] I,
                float[] gravity,
                float[] geomagnetic)
```

Computes the inclination matrix **I** as well as the rotation matrix **R** transforming a vector from the device coordinate system to the world's coordinate system which is defined as a direct orthonormal basis, where:

- X is defined as the vector product **Y.Z** (It is tangential to the ground at the device's current location and roughly points East).
- Y is tangential to the ground at the device's current location and points towards the magnetic North Pole.
- Z points towards the sky and is perpendicular to the ground.



Source: https://developer.android.com/reference/android/hardware/SensorManager.html#getRotationMatrix(float[], float[], float[], float[])

Claim 12

The method of claim 10, wherein the orientation output is a rotation matrix, a **quaternion**, a rotation vector, or comprises **three orientation angles**.

The `getQuaternionFromVector()` function outputs a **quaternion**.

`getQuaternionFromVector(float[] Q, float[] rv)`

Helper function to convert a rotation vector to a normalized quaternion.

The `getOrientation()` function outputs **three orientation angles**.

getOrientation

Added in API level 3

```
float[] getOrientation (float[] R,
                float[] values)
```

Computes the device's orientation based on the rotation matrix.

When it returns, the array values are as follows:

- values[0]: *Azimuth*, angle of rotation about the -z axis. This value represents the angle between the device's y axis and the magnetic north pole. When facing north, this angle is 0, when facing south, this angle is π. Likewise, when facing east, this angle is π/2, and when facing west, this angle is -π/2. The range of values is -π to π.
- values[1]: *Pitch*, angle of rotation about the x axis. This value represents the angle between a plane parallel to the device's screen and a plane parallel to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the top edge of the device toward the ground creates a positive pitch angle. The range of values is -π to π.
- values[2]: *Roll*, angle of rotation about the y axis. This value represents the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the left edge of the device toward the ground creates a positive roll angle. The range of values is -π/2 to π/2.

Source: https://developer.android.com/reference/android/hardware/SensorManager.html#getOrientation(float[], float[])